SE 3XA3: Module Guide
Jumble Words

Team 08, Shunbill Jumble
Shesan Balachandran, balacs1
Bill Nguyen, nguyew3
Muneeb Arshad, arsham14

March 18, 2021

Table 1: **Revision History**

Date	**Version**	**Notes**
March 15 2021	0.0	Initial Draft

Contents

List of Tables

List of Figures

1 Introduction

Decomposing a system into modules is a commonly accepted approach to developing software. A module is a work assignment for a programmer or programming team (Parnas et al., 1984). We advocate a decomposition based on the principle of information hiding (Parnas, 1972). This principle supports design for change, because the "secrets" that each module hides represent likely future changes. Design for change is valuable in SC, where modifications are frequent, especially during initial development as the solution space is explored.

Our design follows the rules layed out by Parnas et al. (1984), as follows:

- System details that are likely to change independently should be the secrets of separate modules.
- Each data structure is used in only one module.
- Any other program that requires information stored in a module's data structures must obtain it by calling access programs belonging to that module.

After completing the first stage of the design, the Software Requirements Specification (SRS), the Module Guide (MG) is developed (Parnas et al., 1984). The MG specifies the modular structure of the system and is intended to allow both designers and maintainers to easily identify the parts of the software. The potential readers of this document are as follows:

- New project members: This document can be a guide for a new project member to easily understand the overall structure and quickly find the relevant modules they are searching for.
- Maintainers: The hierarchical structure of the module guide improves the maintainers' understanding when they need to make changes to the system. It is important for a maintainer to update the relevant sections of the document after changes have been made.
- Designers: Once the module guide has been written, it can be used to check for consistency, feasibility and flexibility. Designers can verify the system in various ways, such as consistency among modules, feasibility of the decomposition, and flexibility of the design.

The rest of the document is organized as follows. Section 2 lists the anticipated and unlikely changes of the software requirements. Section 3 summarizes the module decomposition that was constructed according to the likely changes. Section 4 specifies the connections between the software requirements and the modules. Section 5 gives a detailed description of the modules. Section 6 includes two traceability matrices. One checks the completeness of the design against the requirements provided in the SRS. The other shows the relation between anticipated changes and the modules. Section 7 describes the use relation between modules.

2 Anticipated and Unlikely Changes

This section lists possible changes to the system. According to the likeliness of the change, the possible changes are classified into two categories. Anticipated changes are listed in Section 2.1, and unlikely changes are listed in Section 2.2.

2.1 Anticipated Changes

Anticipated changes are the source of the information that is to be hidden inside the modules. Ideally, changing one of the anticipated changes will only require changing the one module that hides the associated decision. The approach adapted here is called design for change.

AC1: The specific hardware on which the software is running.

AC2: The format of the initial input data.

AC3: The format of the leaderboard display (adding more scores in the leaderboard, change the way the leaderboard is ordered)

AC4: adding more categories for the user to choose from

AC5: The format of difficulty level (currently have difficulty based on length of words can be also lower the amount of time based on difficulty setting)

AC6: The GUIs for the display of the game, main menu and settings

AC7: Currently using external files for data such as words and leaderboard, can change all data storage to an actual database such as postgres

2.2 Unlikely Changes

The module design should be as general as possible. However, a general system is more complex. Sometimes this complexity is not necessary. Fixing some design decisions at the system architecture stage can simplify the software design. If these decision should later need to be changed, then many parts of the design will potentially need to be modified. Hence, it is not intended that these decisions will be changed.

UC1: Input/Output devices (Input: File and/or Keyboard, Output: File, Memory, and/or Screen).

UC2: There will always be a source of input data external to the software.

UC3: There will be an external output file for the leader board so external storage required.

UC4: The game logic for calculating points for leader boards.

3 Module Hierarchy

This section provides an overview of the module design. Modules are summarized in a hierarchy decomposed by secrets in Table 2. The modules listed below, which are leaves in the hierarchy tree, are the modules that will actually be implemented.

M1: Main GUI Module

M2: Settings Module

M3: Settings GUI Module

M4: Game Control Module

M5: User GUI Module

M6: User Module

M7: Leaderboard Module

Level 1	**Level 2**
Hardware-Hiding Module	
Behaviour-Hiding Module	Main GUI Module Settings Module Settings GUI Module Game Control Module User GUI Module
Software Decision Module	User Module Leaderboard Module

Table 2: Module Hierarchy

4 Connection Between Requirements and Design

The design of the system is intended to satisfy the requirements developed in the SRS. In this stage, the system is decomposed into modules. The connection between requirements and modules is listed in Table 3.

The requirements described from the SRS can be described in a way that will allow the user to start/initiate the program, play the game correctly, while also being able to view aspects such as the leaderboard, and also modify the necessary settings such as game mode, difficulty level and categories. The modules that were mentioned in section 3 are

going to be implemented in order to get full functionality for the program. To start the MainGUI Module helps to make certain that user can access the program/game and that the display/functionality of the main menu works as intended. The Settings Module helps make certain that the user is able to configure settings and that the settings (game mode, difficulty level and category) that are configured by the user are set correctly. The GameControl Module helps ensure that the game functionality is working as intended such when guessing a correct word the score updates, when the user wants to return to main menu the user will return to the main menu etc. The User GUI Module helps make certain that the display of the username menu is displayed properly and everything is initialized as intended. The User Module ensures the functionality of the user is working as intended such as when a new user is added it gets added properly, when a new score is recorded it updates the score of the user properly etc. Finally the Leaderboard Module helps make certain that the program will display the correct top ten usernames and scores from highest to lowest and that it gets updated accordingly.

5 Module Decomposition

Modules are decomposed according to the principle of "information hiding" proposed by Parnas et al. (1984). The *Secrets* field in a module decomposition is a brief statement of the design decision hidden by the module. The *Services* field specifies *what* the module will do without documenting *how* to do it. For each module, a suggestion for the implementing software is given under the *Implemented By* title. If the entry is *OS*, this means that the module is provided by the operating system or by standard programming language libraries. Also indicate if the module will be implemented specifically for the software.

Only the leaf modules in the hierarchy have to be implemented. If a dash (–) is shown, this means that the module is not a leaf and will not have to be implemented. Whether or not this module is implemented depends on the programming language selected.

5.1 Hardware Hiding Modules

5.2 Behaviour-Hiding Module

5.2.1 Main GUI Module (M1)

Secrets: Functionalities of the main screen.

Services: Start game, leaderboard and quit game functionalities

Implemented By: [TkInter, Python3]

5.2.2 Settings Module (M2)

Secrets: Manipulate game settings.

Services: Change difficulty, game mode, categories.

Implemented By: [Python3]

5.2.3 Settings GUI Module (M3)

Secrets: User interface elements including buttons and labels

Services: Select game difficulty, game mode, and categories through buttons

Implemented By: [TkInter, Python3]

5.2.4 Game Control Module (M4)

Secrets: Manipulate game related interactions.

Services: Submit answer, guess word, go to next question or go back to main menu.

Implemented By: [Python3]

5.2.5 User GUI Module (M5)

Secrets: Display user related elements.

Services: Input username and submit username

Implemented By: [TkInter, Python3]

5.3 Software Decision Module

5.3.1 User Module (M6)

Secrets: Manages the user information.

Services: Provide functionalities to insert, delete and update user data

Implemented By: [Python3]

5.3.2 Leader board Module (M7)

Secrets: Manage the user high-scores

Services: Provides functionalities to insert, delete and update high scores for different users

Implemented By: [Python3]

6 Traceability Matrix

This section shows two traceability matrices: between the modules and the requirements and between the modules and the anticipated changes.

Req.	**Modules**
FR01	M5, M6
FR02	M2, M3
FR03	M2, M3
FR04	M2, M3
FR05	M1, M7
FR06	M4
FR07	M4
FR08	M4
FR09	M5
FR10	M4
FR11	M1, M7
FR12	M1

Table 3: Trace Between Requirements and Modules

AC	**Modules**
AC1	M2, M3
AC2	M4, M5, M6
AC3	M7
AC4	M2, M3
AC5	M2, M3
AC6	M1, M2, M3, M5, M6
AC7	M2, M4, M7

Table 4: Trace Between Anticipated Changes and Modules

7 Use Hierarchy Between Modules

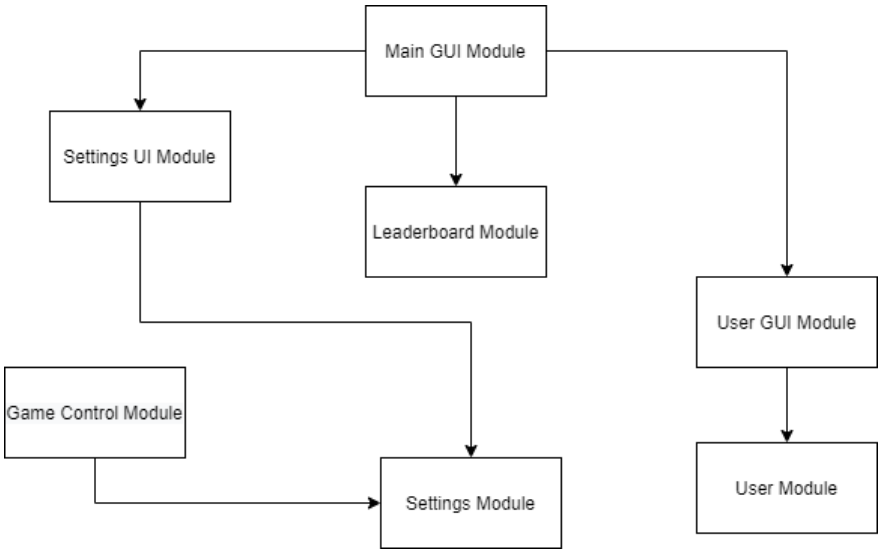

Figure 1: Use hierarchy among modules

References

David L. Parnas. On the criteria to be used in decomposing systems into modules. *Comm. ACM*, 15(2):1053–1058, December 1972.

D.L. Parnas, P.C. Clement, and D. M. Weiss. The modular structure of complex systems. In *International Conference on Software Engineering*, pages 408–419, 1984.